**РЕФЕРАТ**

Отчет 21 с., 8 рис., 2 табл., 4 источника, 1 приложение.

АРИФМЕТИКО-ЛОГИЧЕСКОЕ УСТРОЙСТВО (АЛУ), УПРАВЛЯЮЩЕЕ УСТРОЙСТВО (УУ), СУММИРОВАНИЕ, УМНОЖЕНИЕ, КОМПЬЮТЕРНАЯ АРИФМЕТИКА.

Цель работы – разработать ускоренный умножитель по алгоритму Лемана. Разрядность сомножителей – восемь бит без знака.

При разработке умножителя использовались концепции «черного ящика», т.е. первоначальное определение общих функций устройства и системы входных и выходных сигналов. В основе дальнейшей работы с «черным ящиком» использовался принцип декомпозиции, т.е. последовательное разложение функций на подфункции до получения описания функций на элементарном уровне.

В результате работы была составлен алгоритм работы и структура устройства.

Приведен контрольный пример в числовой форме.

**СОДЕРЖАНИЕ**

[ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМЫХ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СОКРАЩЕНИЙ И ТЕРМИНОВ](#_heading=h.gjdgxs) [6](#_heading=h.gjdgxs)

[ВВЕДЕНИЕ](#_heading=h.30j0zll) [6](#_heading=h.30j0zll)

[1 Исследование предметной области курсовой работы](#_heading=h.2et92p0) [8](#_heading=h.2et92p0)

[1.1 Алгоритм ускоренного умножения](#_heading=h.tyjcwt) [8](#_heading=h.tyjcwt)

[2 Разработка устройства](#_heading=h.3dy6vkm) [10](#_heading=h.3dy6vkm)

[2.1 Анализ исходных данных задания на курсовую работу](#_heading=h.1t3h5sf) [10](#_heading=h.1t3h5sf)

[2.2 Спецификация устройства на уровне «черного ящика»](#_heading=h.4d34og8) [10](#_heading=h.4d34og8)

[2.3 Представление «черного ящика» в виде операционной и управляющей частей](#_heading=h.2s8eyo1) [10](#_heading=h.2s8eyo1)

[2.4 Разработка структуры операционной части устройства](#_heading=h.17dp8vu) [10](#_heading=h.17dp8vu)

[2.5 Разработка схемы алгоритма работы](#_heading=h.26in1rg) [11](#_heading=h.26in1rg)

[2.6 Составление полной спецификации устройства](#_heading=h.lnxbz9) [13](#_heading=h.lnxbz9)

[2.8 Разработка фрагмента функциональной схемы вычитателя](#_heading=h.44sinio) [14](#_heading=h.44sinio)

[2.9 Контрольный пример](#_heading=h.z337ya) [14](#_heading=h.z337ya)

[2.10 Временная диаграмма работы УУ](#_heading=h.1y810tw) [16](#_heading=h.1y810tw)

[ЗАКЛЮЧЕНИЕ](#_heading=h.4i7ojhp) [17](#_heading=h.4i7ojhp)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](#_heading=h.2xcytpi) [18](#_heading=h.2xcytpi)

[ПРИЛОЖЕНИЕ А.](#_heading=h.1ci93xb) 19

# ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМЫХ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СОКРАЩЕНИЙ И ТЕРМИНОВ

| ЧЯ | Черный ящик |
| --- | --- |
| УЧ | Управляющая часть устройства |
| ОЧ | Операционная часть устройства |
| АЛУ | Арифметико-логическое устройство |
| МО | Микрооперация |
| МПР | Микропрограмма |

# ВВЕДЕНИЕ

Цель работы (проекта) – Разработать ускоренный умножитель по алгоритму Лемана. Разрядность сомножителей – 8 бит, без знака.

Объектом исследования является устройство управления. Предметом исследования – алгоритм работы и структура устройства.

В работе представлена спецификация устройства на уровне «черного ящика», разработана схема алгоритма работы устройства и его микропрограммы, составлена полная спецификация устройства. Построена временная диаграмма работы устройства управления. Приведен листинг разработанной программы на языке программирования С++ и результаты расчета контрольного примера.

# 1 Исследование предметной области курсовой работы

## 1.1 Алгоритм ускоренного умножения

Метод Лемана рассмотрим в предположении, что умножение выполняется, начиная с младших разрядов множителя.

Алгоритм в данном цикле операции умножения может быть записан в следующем виде:

,

где: – номер разряда множителя;

– цифра -го разряда множителя;

– двоичная переменная, единичное значение которой для соответствующего разряда множителя указывает на необходимость выполнения арифметического действия между суммой частичных произведений и множимым (сложение или вычитание);

– знак арифметического действия.

Очевидно, что при .

Если , то производится сложение суммы частичных произведений с множимым. При производится вычитание множимого из суммы частичных произведений.

Из анализа приведенных выше выражений, описывающих алгоритм действий в данном цикле умножения, следует, что перед началом операции умножения . Это значение сохраняется до появления первой единицы в младшем разряде множителя. При появлении там единицы производится сложение множимого с предшествующей суммой частичных произведений, если в следующем по старшинству разряде множителя содержится «0», или производится вычитание множимого из предшествующей суммы частичных произведений, если в следующем по старшинству разряде множителя содержится «1», то есть .

При появлении 0 в младшем разряде множителя производится вычитание множимого из предшествующей суммы частичных произведений, если в следующем по старшинству разряде множителя содержится «1» , или производится сложение множимого с предшествующей суммой частичных произведений, если в следующем по старшинству разряде множителя содержится «0» .

Рассмотрим на конкретном примере последовательность действий в сумматоре и регистре множителя при умножении по методу Лемана.

Пусть множимое , а множитель .

Будем считать, что множитель и сумма частичных произведений в каждом цикле вычислений сдвигаются на один разряд вправо соответственно в регистре множителя и в сумматоре. Сложение (вычитание) в сумматоре производится в обратном модифицированном коде. Для округления результата в сумматоре имеется дополнительный разряд справа.

С учетом сказанного схема выполнения конкретного примера ускоренного умножения по методу Лемана представлена в таблице 1.

ТАБЛИЦА 1 – Схема выполнения примера ускоренного умножения

|  | | |  | |
| --- | --- | --- | --- | --- |
| Содержимое регистра |  |  | Содержимое сумматора | |
| 100011 | 10 | исходное состояние |  | 00 00000000 0 |
| 010001 | 11 | сдвиг вправо | и | 00 00000000 0 |
|  |  | (-) | + | 11 01001011 1 |
|  |  |  |  | 11 01001011 1 |
| 001000 | 11 | сдвиг вправо | и | 11 10100101 1 |
| 000100 | 01 | сдвиг вправо | и | 11 11010010 1 |
| 000010 | 00 | сдвиг вправо | и | 11 11101001 0 |
|  |  | (+) | + | 00 10110100 0 |
|  |  |  | 1 | 00 10011101 0 |
|  |  |  |  | 00 10011101 1 |
| 000001 | 00 | сдвиг вправо | и | 00 01001110 1 |
| 000000 | 10 | сдвиг вправо | и | 00 00100111 0 |
| 000000 | 01 | сдвиг вправо | и | 00 00010011 1 |
|  |  | (+) | + | 00 10110100 0 |
|  |  | Произведение Z = X\*Y |  | 00 11000111 1 |
|  |  | Округление Z |  | 00 11001000 0 |
|  | | | | |

Непосредственным умножением можно легко проверить, что полученное значение произведения соответствует истинному с учетом округления.

Анализ показывает, что при умножении по методу Лемана даже при наиболее неблагоприятном сочетании цифр *n* разрядного множителя (101010…) количество операций суммирования равно *n/2*. В среднем же количество операций суммирования равно *n/3.*

# 2 Разработка устройства

## 2.1 Анализ исходных данных задания на курсовую работу

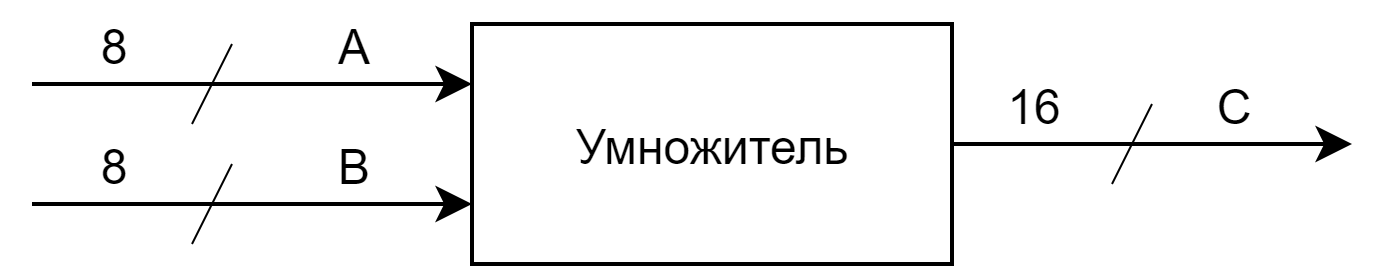
Согласно заданию, устройство должно быть предназначено для выполнения операции “умножение”.

Разрядность входных данных – 8 бит.

Исходя из этого очевидно, что входы (А,В) должны иметь 8 разрядов, а выход (результат операции, С) должен иметь 16 разрядов (максимальное количество разрядов при умножении двух восьмиразрядных чисел)

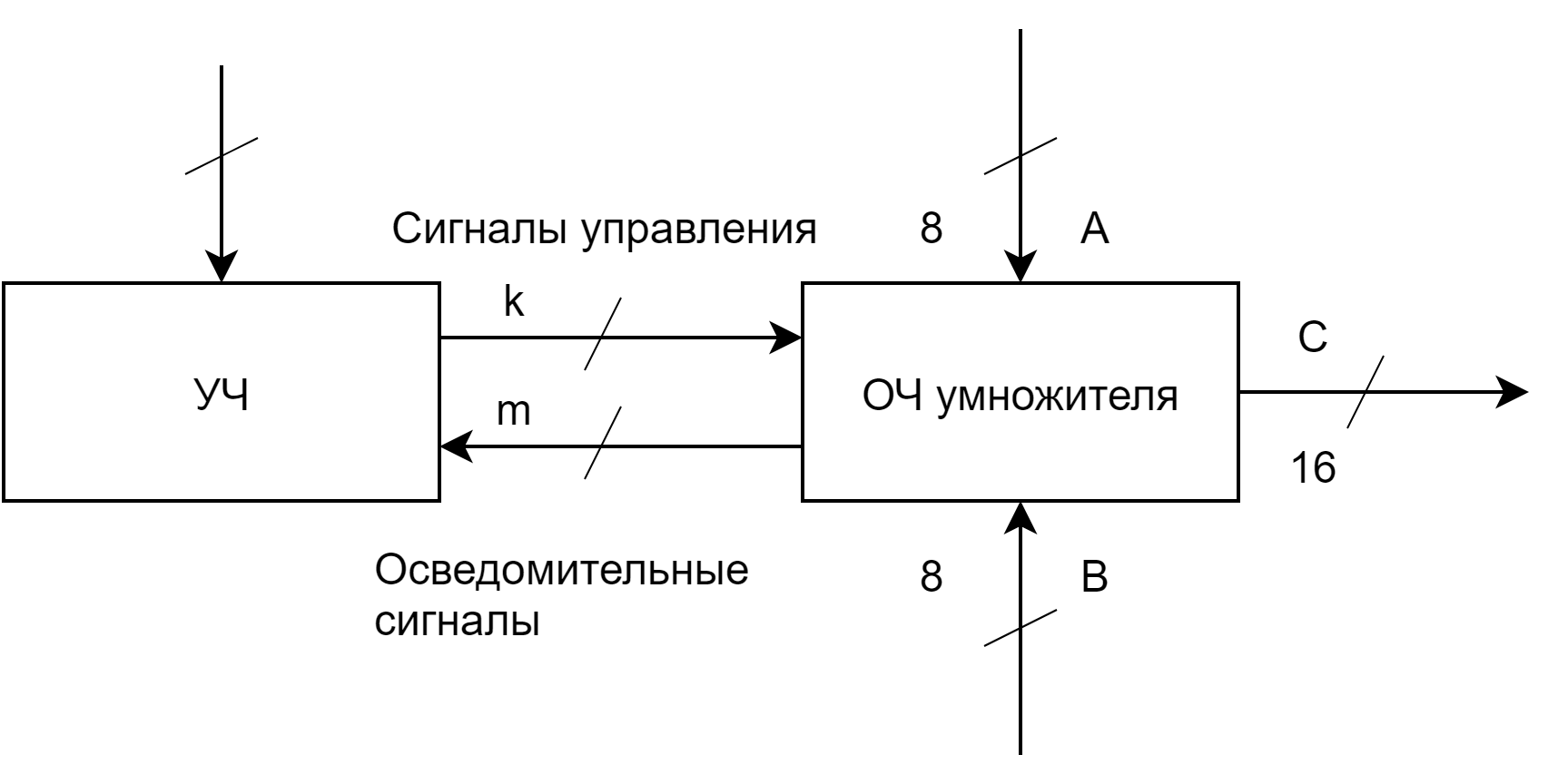
## 2.2 Спецификация устройства на уровне «черного ящика»

В задании сказано, что устройство предназначено для выполнения операции умножения чисел, поэтому должно быть два 11– ти разрядных входа А и В для уменьшаемого и один 21– ти разрядный выход С для результата.

[](https://app.diagrams.net/?page-id=06Kz00Qnp5J-cYQDDaIb&scale=auto#G1AxRg_vjvCM4gIGCwZyi0PaIKqb1hakvw)  
Рисунок 1 – Система выводов устройства “Умножитель”

## 2.3 Представление «черного ящика» в виде операционной и управляющей частей

Упрощенно устройство разрабатываемого умножителя можно представить схемой из рисунка 2.

[](https://app.diagrams.net/?page-id=BXWEEg_pVuuhafz6vGLZ&scale=auto#G1AxRg_vjvCM4gIGCwZyi0PaIKqb1hakvw)  
Рисунок 2 – Разбиение схемы устройств на ОЧ и УЧ.

## 2.4 Разработка структуры операционной части устройства

Пусть операнды размещаются в регистре А (множиоме) и в регистре В (множитель). Алгоритм приведен на рисунке 3.

Алгоритм не учитывает микрооперации, в пользовательском интерфейсе можно добавить возможность включения пошагового вывода работы умножителя.

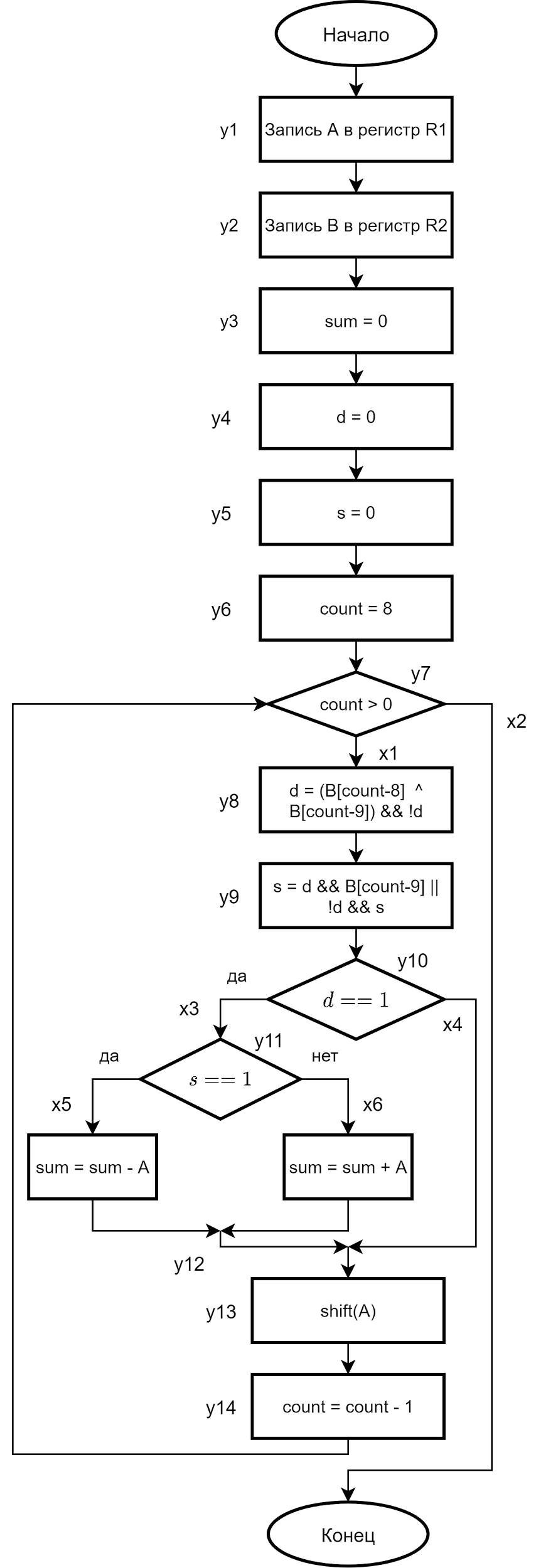
[](https://app.diagrams.net/?page-id=1ocrjFJL5byTHJxX0K9V&scale=auto#G1AxRg_vjvCM4gIGCwZyi0PaIKqb1hakvw)

Рисунок 3 – Упрощённая структура ОЧ умножителя.

## 

## 2.5 Разработка схемы алгоритма работы

Схема алгоритма на уровне микроопераций изображена на рисунке 4

[](https://app.diagrams.net/?page-id=-IDVpmjxW6K69LzNgD0H&scale=auto#G1AxRg_vjvCM4gIGCwZyi0PaIKqb1hakvw)  
Рисунок 4 – Схема алгоритма на уровне микроопераций

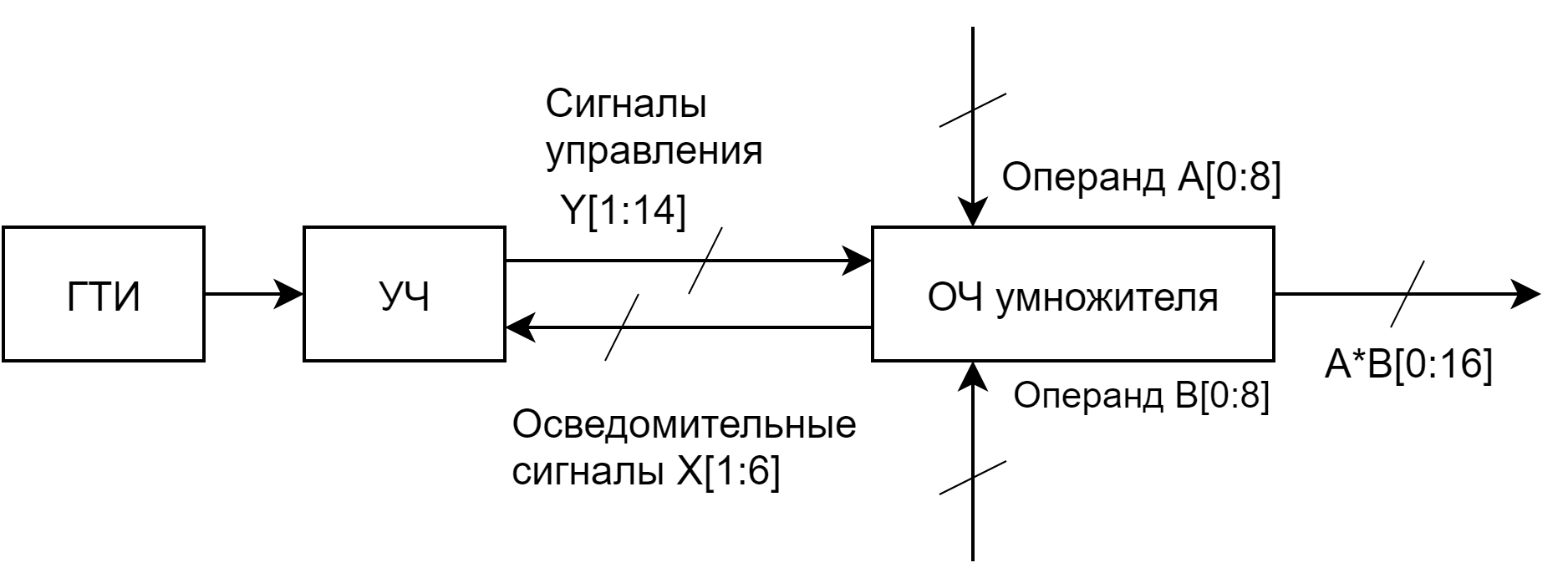
## 2.6 Составление полной спецификации устройства

В таблице 2 приведено описание всех линий и сигналов, полученных в процессе разработки умножителя.

ТАБЛИЦА 2 – Сигналы умножителя

| Имя сигнала/шины  и разрядность | Тип  (In/Out) | Назначение сигнала |
| --- | --- | --- |
| Y1 | I для ОЧ | Сигнал о записи множимого в регистр R1 |
| Y2 | I для ОЧ | Сигнал о запись множителя в регистр R2 |
| Y3 | I для ОЧ | Сигнал о записи 0 в регистр суммы в R3 |
| Y4 | I для ОЧ | Сигнал об установке значения 0 для флага d |
| Y5 | I для ОЧ | Сигнал об установке значения 0 для флага s |
| Y6 | I для ОЧ | Сигнал о записи 8 в счётчик |
| Y7 | I для ОЧ | Сигнал о начале цикла |
| Y8 | I для ОЧ | Сигнал об установке флага d |
| Y9 | I для ОЧ | Сигнал об установке флага s |
| Y10 | I для ОЧ | Сигнал о проверке установки флага d |
| Y11 | I для ОЧ | Сигнал о проверке установки флага s |
| Y12 | I для ОЧ | Сигнал об окончании ветвления |
| Y13 | I для ОЧ | Сигнал о сдвиге регистра R1 влево на бит |
| Y14 | I для ОЧ | Сигнал об уменьшении счётчика на 1 |
| X1 | O для ОЧ | Сигнал о шаге цикла |
| X2 | O для ОЧ | Сигнал об окончании цикла |
| X3 | O для ОЧ | Сигнал о том, что флаг d установлен (необходимо произвести операцию) |
| X4 | O для ОЧ | Сигнал о том, что флаг d не установлен |
| X5 | O для ОЧ | Сигнал о том, что флаг s установлен (произведено вычитание) |
| X6 | O для ОЧ | Сигнал о том, что флаг s не установлен (произведено сложение) |

Структура умножителя (фрагмента программы) на микропрограммном уровне управления приведена на рисунке 5.

[](https://app.diagrams.net/?page-id=RJgZOfOWHPpLQjlEorSh&scale=auto#G1AxRg_vjvCM4gIGCwZyi0PaIKqb1hakvw)  
Рисунок 5 – Структура вычитателя на микропрограммном уровне управления

## 2.8 Разработка фрагмента функциональной схемы вычитателя

Фрагмент схемы УЧ дан на рисунке 6.

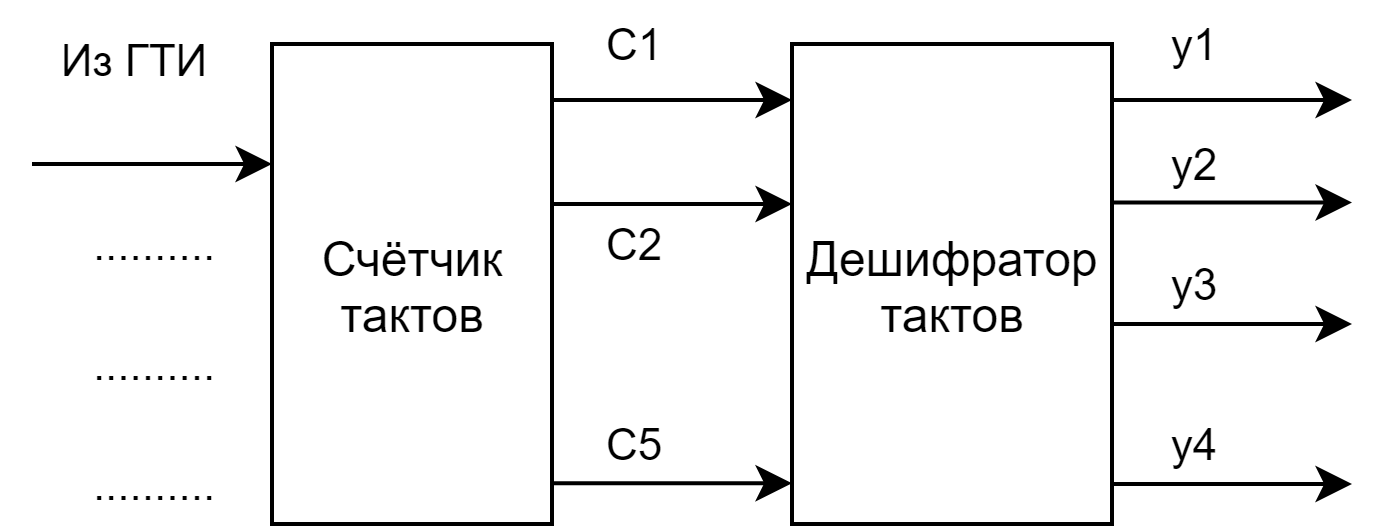
[](https://app.diagrams.net/?page-id=Iaj6JvLA2qxfLnv7LyFe&scale=auto#G1AxRg_vjvCM4gIGCwZyi0PaIKqb1hakvw)

Рисунок 6 – Фрагмент схемы УЧ

## 

## 2.9 Контрольный пример

Исходные данные:

A = 246

B = 23

Листинг программы:

| #include <iostream>  **int** lemansMethod(**int**, **int**, **bool** = **false**);  **int** main(**int**, **char**\*\*) {  **int** a, b;  **char** sbs;  **do** {  std::cout << **"Insert a, 0 <= a < 256: "**; std::cin >> a;  } **while** (a < 0 || a > 255);  **do** {  std::cout << **"Insert b, 0 <= b < 256: "**; std::cin >> b;  } **while** (b < 0 || b > 255);  std::cout << **"step-by-step solution, Y/n: "**; std::cin >> sbs;   std::cout << **"Leman's method solution: "** << lemansMethod(a, b, sbs == **'Y'**); } |
| --- |

Исходный код ускоренного умножения по методу Лемана дан в приложении А.

Выполнение программы для исходных данных показано на рисунке 7.

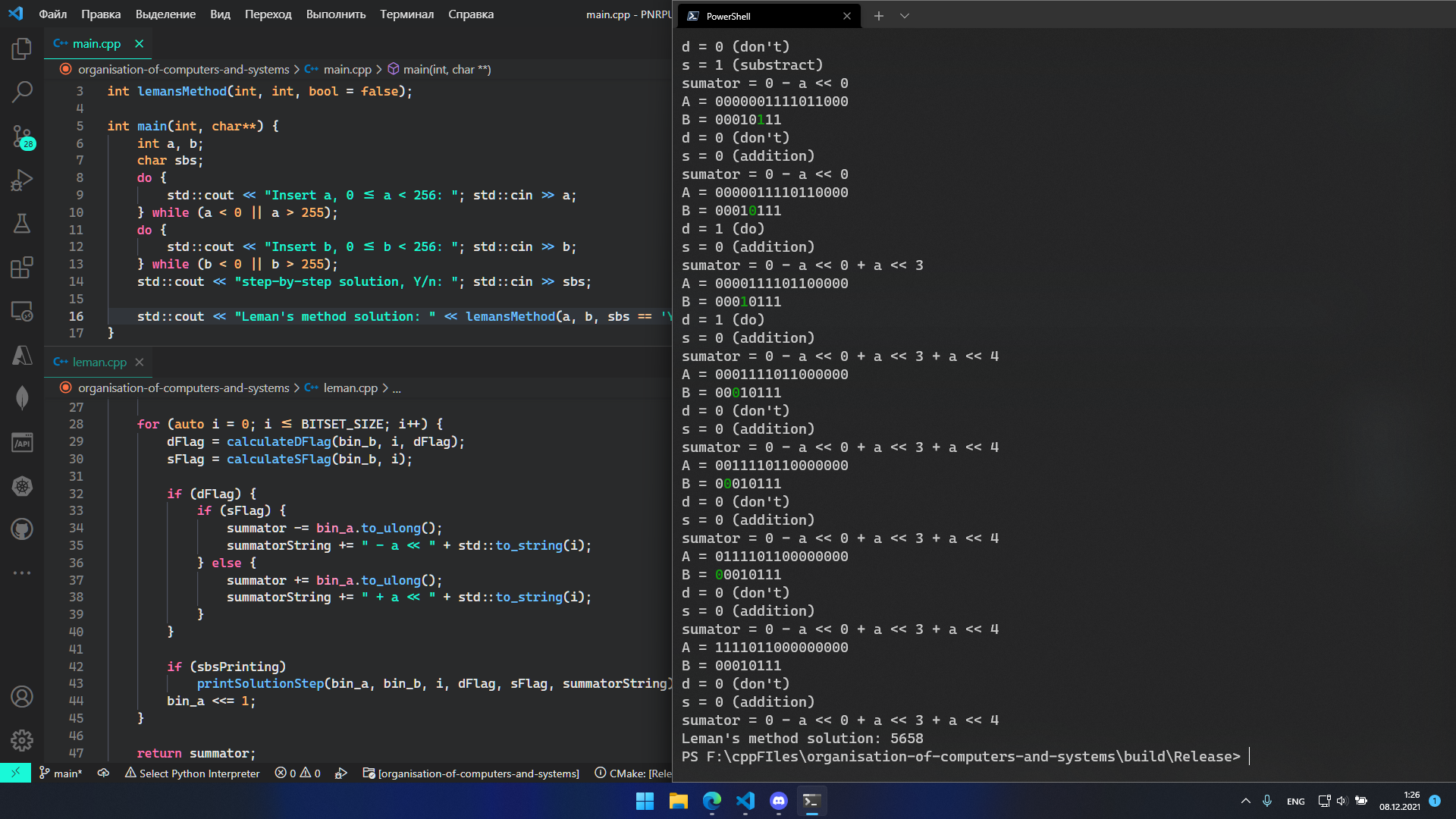


Рисунок 7 – Выполнение программы

## 2.10 Временная диаграмма работы УУ

На рисунке 8 приведен фрагмент временной диаграммы работы ОЧ

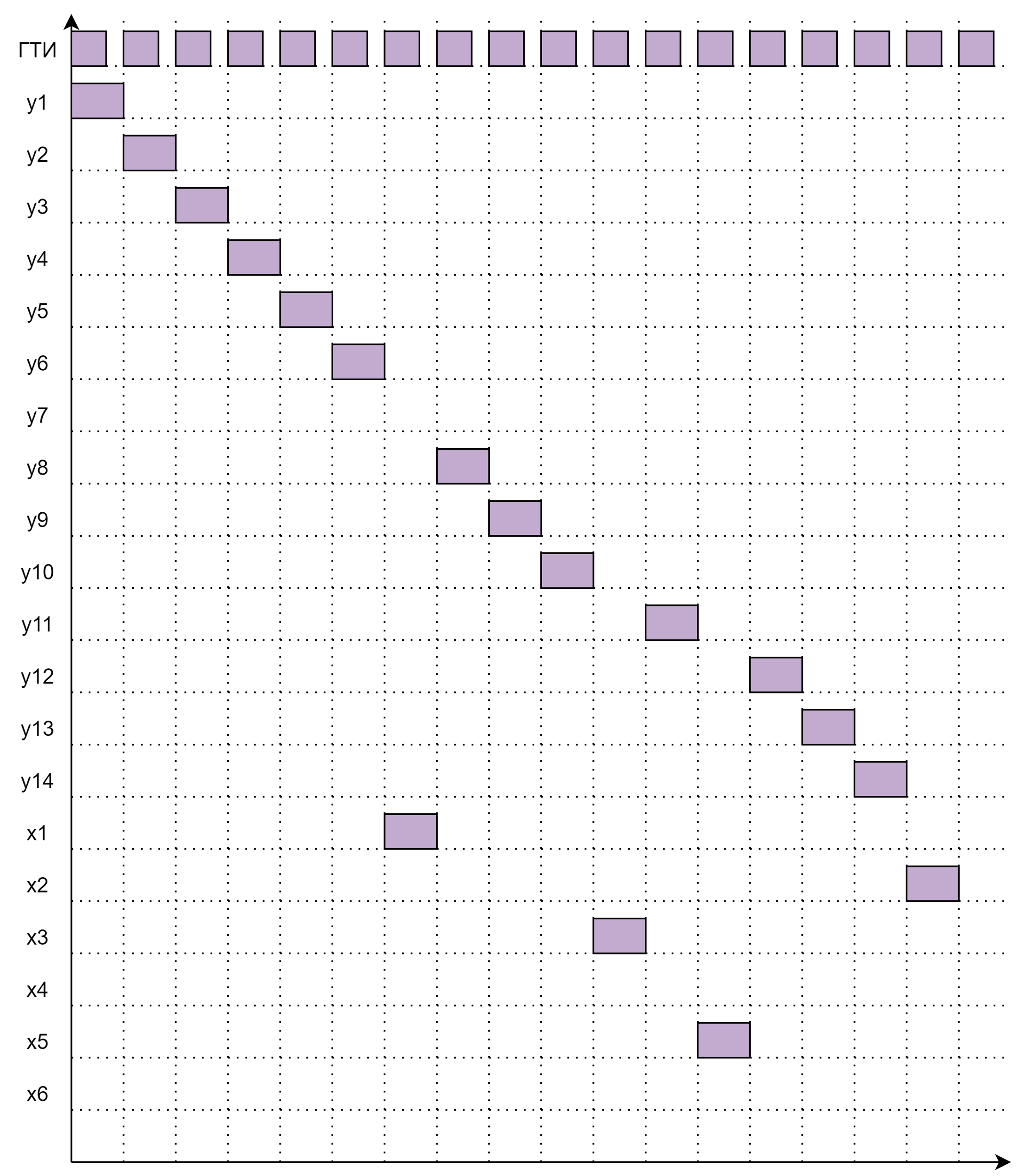
[](https://app.diagrams.net/?page-id=rx2Oe3KUFnc6ynS12FHu&scale=auto#G1AxRg_vjvCM4gIGCwZyi0PaIKqb1hakvw)

Рисунок 8 – Фрагмент временной диаграммы работы умножителя

Закрашенные интервалы времени соответствуют логическим 1, горизонтальные штриховые линии указывают интервалы времени, в которых значения X не имеют смысла, т.к. в эти интервалы сигналы X не проверяются в УЧ. Вертикальные штриховые линии разделяют временные такты.

# ЗАКЛЮЧЕНИЕ

Задача курсовой работы – разработка ускоренного умножителя по алгоритму Лемана.

Поставленная задача выполнена. В ходе курсовой работы была изучена специальная литература, разработана структура ОЧ, алгоритм их работы, спецификация сигналов, фрагмент функциональной схемы УЧ, контрольный числовой пример и временная диаграмма работы устройства.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Павловская, Т.А. С/С++. Программирование на языке высокого уровня: Учеб. пособие. – СПб.:Питер, 2007. – 461 с.
2. Жмакин, А. П. Архитектура ЭВМ: 2-е изд., перераб. и доп.: учеб. пособие. — СПб.: БХВ-Петербург, 2010. — 352 с.
3. Потапов, В.И., Шафеева, О.П., Червенчук, И.В. Основы компьютерной арифметики и логики: Учеб. пособие. – Омск: Изд- во ОмГТУ, 2004. – 172 с
4. Потапов, И. В. Элементы прикладной теории цифровых автоматов: учеб. пособие / И. В. Потапов. – Омск: Изд-во ОмГТУ, 2011. – 156 с

## ПРИЛОЖЕНИЕ А.

**Исходный код ускоренного умножителя по методу Лемана.**

| #include <bitset> #include <iostream> #include <string>  #define NC "\033[0m"; *// Определение цвета вывода обычного текста* #define ACTIVE "\033[32m" *// Определение цвета вывода анализируемого бита*  **const** **size\_t** BITSET\_SIZE = 8; *// Разрядность чисел*  **void** printSolutionStep( *// Печать текущего шага решения*  std::bitset<BITSET\_SIZE\*2> a, *// Двоичная запись числа А*  std::bitset<BITSET\_SIZE> b, *// Двоичная запись числа B*  **int** currentBit, *// Номер анализируемого бита*  **bool** dFlag, *// Флаг d, нужна ли операция*  **bool** sFlag, *// Флаг s, какая операция нужна*  std::string summatorString *// Строка отображения сдвигов* ); **void** printPrepairingsStep( *// Печать этапа подготовки*  std::bitset<BITSET\_SIZE\*2> a, *// Двоичная запись числа A*  std::bitset<BITSET\_SIZE> b *// Двоичная запись числа B* );  **bool** calculateSFlag( *// Функция вычисления флага s*  std::bitset<BITSET\_SIZE> b, *// Двоичная запись числа B*  **int** currentBit *// Номер анализируемого бита* ); **bool** calculateDFlag( *// Функция вычисления флага d*  std::bitset<BITSET\_SIZE> b, *// Двоочная запись числа b*  **int** currentBit, *// Номер анализирумеого бита*  **bool** previousDFlag *// Предыдущее значение флага d* );  **int** lemansMethod(**int** a, **int** b, **bool** sbsPrinting = **false**) {  std::bitset<BITSET\_SIZE\*2> bin\_a(a);  std::bitset<BITSET\_SIZE> bin\_b(b);   **bool** dFlag = **false**, sFlag = **false**;   **int** summator = 0;  std::string summatorString = **"0"**;   **if** (sbsPrinting)  printPrepairingsStep(bin\_a, bin\_b);   **for** (**auto** i = 0; i <= BITSET\_SIZE; i++) {  dFlag = calculateDFlag(bin\_b, i, dFlag);  sFlag = calculateSFlag(bin\_b, i);   **if** (dFlag) {  **if** (sFlag) {  summator -= bin\_a.to\_ulong();  summatorString += **" - a << "** + std::to\_string(i);  } **else** {  summator += bin\_a.to\_ulong();  summatorString += **" + a << "** + std::to\_string(i);  }  }   **if** (sbsPrinting)  printSolutionStep(bin\_a, bin\_b, i, dFlag, sFlag, summatorString);  bin\_a <<= 1;  }   **return** summator; }  **void** printPrepairingsStep(std::bitset<BITSET\_SIZE\*2> a, std::bitset<BITSET\_SIZE> b) {  std::cout <<   **"A = "** << a.to\_string() << **'\n'** <<  **"B = "** << b.to\_string() << **'\n'** <<  **"d = "** << 0 << **'\n'** <<  **"s = "** << 0 << **'\n'**;  }  **void** printSolutionStep(std::bitset<BITSET\_SIZE\*2> a, std::bitset<BITSET\_SIZE> b, **int** currentBit, **bool** dFlag, **bool** sFlag, std::string summatorString) {  std::cout << **"A = "** << a.to\_string() << **'\n'** << **"B = "**;  **for** (**auto** i = 0; i < BITSET\_SIZE; i++) {  **if** (BITSET\_SIZE - i - 1 == currentBit) std::cout << ACTIVE << b[BITSET\_SIZE - i - 1] << NC  **else** std::cout << b[BITSET\_SIZE - i - 1];  }  std::cout << **'\n'** <<  **"d = "** << dFlag << **" "** << (dFlag ? **"(do)"** : **"(don't)"**) << **'\n'** <<  **"s = "** << sFlag << **" "** << (sFlag ? **"(substract)"** : **"(addition)"**) << **'\n'** <<  **"sumator = "** << summatorString << **'\n'**; }  **bool** calculateSFlag(std::bitset<BITSET\_SIZE> b, **int** currentBit) {  **if** (currentBit + 1 >= BITSET\_SIZE) **return** **false**;  **return** b.test(currentBit) && b.test(currentBit+1); }  **bool** calculateDFlag(std::bitset<BITSET\_SIZE> b, **int** currentBit, **bool** previousDFlag) {  **if** (currentBit == 0)  **return** b.test(currentBit);  **else** **if** (currentBit == BITSET\_SIZE)  **return** b.test(currentBit-1) && !previousDFlag;  **else**  **return** (b.test(currentBit) && !b.test(currentBit-1)) || (!b.test(currentBit) && b.test(currentBit-1) && !previousDFlag); } |
| --- |